Practical Machine Project - Predicting the type of barbell lifts performed

Fariz Abdul Rahman

August 14, 2016

Overview

The goal of the project is to predict the manner in which people performed barbell lifts correctly and incorrectly, in a total of 5 different ways. This report describes how the model is built with explanation of the choices made and how the model was cross validated by examining its out-of-sample error rate. The model will be applied on 20 test cases to examine its accuracy.

Method

A training and testing database is provided from data collected using sensors to record participants performing the barbell lifts in 5 different ways categorized as *classe* in the dataset. The output of this work is a prediction model for the 5 different barbell lift motions which has a low out-of-sample error rate. The tasks that will be performed are:

1. Obtaining the data and tidying it up.
2. Partitioning the data for cross validation.
3. Building the prediction model using random forest.
4. Analyzing the out-of-sample error.
5. Applying the prediction model to the testing dataset.

Task 1: Tidying the training data

The training and testing data is obtained from the internet and read into R:

```
setwd("~//R//practical machine learning project")
url_1 <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
download.file(url_1, destfile="pml-training.csv")
url_2 <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
download.file(url_2, destfile="pml-testing.csv")
raw_training <- read.csv("pml-training.csv", na.strings = c("","NA","#DIV/0!"))
raw_testing <- read.csv("pml-testing.csv", na.strings = c("","NA","#DIV/0!"))
```

Examining the raw training dataset with shows 19622 observations of 160 variables. Performing 60, 67, 1, 1, 1, 4, 1, 4, 2, 2, 1, 1, 2, 1, 4, 2, 6 shows that 60 variables have complete data and 100 variables have 19216 *NA* values. Removing variables with incomplete data and the first seven variables not related to the motion of the barbell lifts reduces the number of variables to predict with to 53. The same variables are removed in the raw testing dataset.

```
training_tidy <- raw_training[,!is.na(raw_training[1,])]
training_tidy <- training_tidy[,-c(1:7)]
length(training_tidy)
```

```
## [1] 53
```

```
testing_tidy <- raw_testing[,(names(raw_testing) %in% colnames(training_tidy))]
```

Task 2: Partitioning the data for cross-validation

Although there is no need for externally cross-validating the random forest model since CARET performs it internally during the model optimization, we can practice doing cross-validation by partitioning the tidy training dataset into 70% for training and 30% for cross validation based on the outcome variable *classe*. A seed is used for reproducibility of results during the data partitioning.

```
set.seed(1234)
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.3.1
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
inTrain <- createDataPartition(training_tidy$classe, p=0.70, list = FALSE)
training <- training_tidy[inTrain,]
testing <- training_tidy[-inTrain,]
```

Task 3: Building the model

The random forest method is selected for this work based on its superior performance as a classification tree among other algorithms. It is highly accurate and does not overfit the data by performing cross validation internally and improving the running error estimates. It computes proximities between pairs of cases that can be used in clustering and locating outliers. It runs efficiently on large databases and allows generated forests to be saved for future use. These features among other features makes it a good choice to analyze our large and varied datasets.

We make use of the feature to run the random forest algorithm in parallel and saving the generated trees for future runs to handle the large datasets when building our model.

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.3.1
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(doParallel)
```

```
## Warning: package 'doParallel' was built under R version 3.3.1
```

```
## Loading required package: foreach
```

```
## Warning: package 'foreach' was built under R version 3.3.1
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
model <- "modFit.Rdata"
if (!file.exists(model)) {
    # If a previous tree is not available, use parallel features and save the generated
    # tree for future use.
    require(parallel)
    require(doParallel)
    clus <- makeCluster(detectCores()-1)
    registerDoParallel(clus)

    modFit_RF <- train(classe ~., data=training, method="rf")
    save(modFit_RF,file="modFit.Rdata")

    stopCluster(clus)
} else {
    # If a previous tree is available, then just load it
    load(file="modFit.Rdata")
}
show(modFit_RF)
```

```
## Random Forest
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 13737, 13737, 13737, 13737, 13737, 13737, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9889451  0.9860187
##   27    0.9884481  0.9853911
##   52    0.9820583  0.9773108
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

Task 4: Analyzing the out-of-sample error

The prediction model is cross validated by applying it to the partitioned testing dataset to analyze its out-of-sample error.

```
pred_RF <- predict(modFit_RF,testing)
confusionMatrix(pred_RF, testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1673    5    0    0    0
##          B    1 1134    2    0    0
##          C    0    0 1024    1    1
##          D    0    0    0  962    1
##          E    0    0    0    1 1080
##
## Overall Statistics
##
##                Accuracy : 0.998
##                  95% CI : (0.9964, 0.9989)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9974
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9994   0.9956   0.9981   0.9979   0.9982
## Specificity            0.9988   0.9994   0.9996   0.9998   0.9998
## Pos Pred Value         0.9970   0.9974   0.9981   0.9990   0.9991
## Neg Pred Value         0.9998   0.9989   0.9996   0.9996   0.9996
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2843   0.1927   0.1740   0.1635   0.1835
## Detection Prevalence   0.2851   0.1932   0.1743   0.1636   0.1837
## Balanced Accuracy      0.9991   0.9975   0.9988   0.9989   0.9990
```

From the cross-validation results, the model achieved a high accuracy score of 99.8% and error rate of 0.2%. This is great results and validates the selection of our prediction model using random forest.

Task 5: Applying the prediction model to the provided raw testing dataset

The prediction model built is applied to the 20 test cases present in the raw testing dataset and printed.

```
predict(modFit_RF, testing_tidy)
```

```
##  [1] B A B A A E D B A A B C B A E E A B B B
## Levels: A B C D E
```

Conclusion

The results from the final prediction on the raw testing dataset was validated in the quiz and received all correct answers.